Evolutionary Inference of Attribute-based Access Control Policies

Eric Medvet[1], Alberto Bartoli[1], Barbara Carminati[2], and Elena Ferrari[2]

[1] Dip. di Ingegneria e Architettura, Università degli Studi di Trieste, Italy
[2] Dip. di Scienze Teoriche e Applicate, Università degli Studi dell'Insubria, Italy

Abstract. The interest in attribute-based access control policies is increasingly growing due to their ability to accommodate the complex security requirements of modern computer systems. With this novel paradigm, access control policies consist of attribute expressions which implicitly describe the properties of subjects and protection objects and which must be satisfied for a request to be allowed. Since specifying a policy in this framework may be very complex, approaches for policy mining, i.e., for inferring a specification automatically from examples in the form of logs of authorized and denied requests, have been recently proposed.
In this work, we propose a multi-objective evolutionary approach for solving the policy mining task. We designed and implemented a problem representation suitable for evolutionary computation, along with several search-optimizing features which have proven to be highly useful in this context: a strategy for learning a policy by learning single rules, each one focused on a subset of requests; a custom initialization of the population; a scheme for diversity promotion and for early termination. We show that our approach deals successfully with case studies of realistic complexity.

1 Introduction

Data are today one of the most strategic asset of any company and organization and, as such, their protection from any kind of improper modifications or unauthorized disclosures is a fundamental service to be provided by any Data Management System. In a data management system, accesses are regulated through access control policies [1] that are then encoded into a set of authorizations and checked by the reference monitor, a trusted software module in charge of enforcing access control. Since the 1970s, several access control models for policy specification have been proposed, including Discretionary Access Control (DAC), Mandatory Access Control (MAC), and Role-based Access Control (RBAC). The common characteristic of these models is that they are identity-based, that is, access control is based on the identity of subjects and protection objects. These models are not scalable and flexible and thus they do not fit very well in the current scenario, characterized by open and distributed systems. Due to these limitations the new Attribute-based Access Control (ABAC) paradigm has recently emerged [2]. The main advantage of ABAC is that the access control process is not identity-based, rather it exploits attributes of the requestor and

resource (e.g., the age of the requestor). Attribute expressions are then used to implicitly denote the sets of users and resources to which a policy applies (e.g., a nurse can add an item in a HR for a patient in the ward in which he/she works). Clearly, the main advantage of ABAC is in terms of flexibility in the specification of protection requirements. In contrast, the drawback is that policy specification becomes more complex and can result in an expensive and time consuming task.

A promising approach to diminish the burden of policy specification is represented by *policy mining*, whose goal is to partially or totally automate the construction of an ABAC policy from available access control information (e.g., access control logs, RBAC policies). Therefore, in this paper, we propose a multi-objective evolutionary approach for learning ABAC policies from sets of authorized and denied access requests. The approach is multi-objective because it aims at learning a policy which, at the same time, is consistent with the input requests, exhibits low complexity and does not use those attributes which uniquely represent user and resource identities, hence exploiting the true potential of the ABAC paradigm.

The evolutionary approach here proposed includes several contributions: (i) a domain-specific phenotypic representation, along with a set of custom genetic operators, which allow individuals to represent valid policy rules in the ABAC paradigm; (ii) an incremental strategy for learning a policy by learning single rules, each one fitting a subset of requests—a form of separate-and-conquer; (iii) a custom initialization of the population; (iv) a diversity promotion scheme; and (v) an early termination criterion.

We experimentally evaluated our proposal on a set of realistic case studies and found that it is always able to obtain a policy which meets the objectives. We also assessed our solution in case of incomplete input—i.e., when the input requests do not fully represent the access control information—and found that it is robust to missing information rates up to 50%.

We would like to remark that the approach presented in this paper can also be a valuable contribution in other security domains, besides policy mining, such as the strategic one of emergency management. Indeed, one of the most widely used approaches to deal with the information needs arising during emergency situations is the Break-the-Glass (BtG) paradigm [3], which allows users to override access control decisions on demand by logging their accesses. The main issue with BtG models is that they could bring the system to an unsafe state due to abuse of BtG policies. An alternative and more secure way to deal with emergency management, which has been recently proposed in [4], is to use a policy-based approach, according to which a set of emergency policies are specified, overriding regular ones during emergency situations. By properly extending the approach presented in this paper, in terms of objectives to be met, emergency policies can be learned from the BtG logs.

2 Related work

The problem of deriving new ABAC policies from access request logs has been first and only investigated in [5] (extended by [6]). The authors propose an algorithm which incorporates some heuristics aimed at merging and simplifying single rules. In this paper, we use the same ABAC language and the same case studies of the cited papers: our method exhibits the same high effectiveness of the method in [6, 5], which is not evolutionary. We think that our proposal could be easier to extend—by incorporating new objectives to be met—in order to fit more specific needs of similar scenarios, such as emergency policy learning from BtG logs.

Other non-evolutionary approaches have been proposed for mining policies from logs for less expressive access control models (e.g., RBAC [7, 8]). In some cases, additional information, besides the request logs, is needed as training data [9].

Usage of evolutionary techniques for inferring RBAC rules explaining the observed actions in environments with tree-structured role hierarchies was proposed in [10]. The aim of the proposal was using the inferred rules for identifying mismatches between user roles and actual processes, as a tool for insider threat detection. No actual assessment was provided. An exercise in security policy inference through evolutionary techniques was proposed in [11]. This work considered rules based on boolean expressions constructed in a simple language and applied Genetic Programming for discovering a single expression capturing all the rules provided as examples. The case studies were composed of very few examples and were mainly a proof-of-concept demonstrating the feasibility of policy inference by means of Genetic Programming. We consider instead a full-fledged security policy language capable of expressing attribute-based rules, and demonstrate that our approach can indeed be applied successfully on realistic testbeds.

An evolutionary framework for learning security policies which need to be updated dynamically is proposed in [12]. This work introduces a stochastic risk-based security policy model based on a few numerical or boolean variables and uses this model for generating examples of access control decisions. These examples are then used for driving a Genetic Programming search aimed at inferring a formula leading to the same decisions as those in the examples. The cited work uses the SPEA2 multi objective evolutionary algorithm [13] for minimizing the error rate and the size of each formula. Evolutionary multi-objective optimization techniques have been applied in other security-related problems as a tool for systematically coping with problem-specific constraints, e.g., performance and usability in network reconfiguration strategies [14] and run-time efficiency in deep packet inspection [15].

3 Scenario

3.1 ABAC policy language

We consider the ABAC policy language defined in [5], which is stated to be, according to the authors, significantly more complex than policy languages handled in previous work on security policy mining. We briefly describe the language below in order to provide the appropriate context for our work.

Let U be a set of *users* and A_U a set of *user attributes*. The *value* of attribute $a \in A_U$ for user $u \in U$ is represented by a function $d_U(u, a)$. This function can assume a special value $\perp$ to indicate that the value of attribute a for user u is undefined.

The set of user attributes A_U can be partitioned in two sets: $A_{U,\mathbb{1}}$, containing *single-valued* attributes, and $A_{U,\infty}$ containing *multi-valued* attributes, i.e., attributes whose values are sets of single values. Set $A_{U,\mathbb{1}}$ includes a special attribute uid which has a unique value (different from $\perp$) for each user.

We denote with $V_U(a)$ the set of possible single values assumed by user attribute a, i.e., the range of d_U for $a \in A_{U,\mathbb{1}}$ and the union of the range elements for $a \in A_{U,\infty}$.

Similarly, let R be a set of *resources* and A_R a set of *resource attributes*. The value of attribute $a \in A_R$ for resource $r \in R$ is represented by a function $d_R(r, a)$, which can assume a special value $\perp$ to indicate that the value of attribute a for resource r is undefined. The set A_R can be partitioned in two sets $A_{R,\mathbb{1}}$ and $A_{R,\infty}$ containing single-valued and multi-valued attributes, respectively. Set $A_{R,\mathbb{1}}$ includes a special attribute rid which has a unique value (different from $\perp$) for each resource. We denote with $V_R(a)$ the set of possible single values assumed by resource attribute a.

Subsets of users and resources can be described by means of *attribute expressions*, as follows. We denote by $\text{Set}(S)$ the powerset of set S. A user attribute expression is a function $e_U : A_U \to E$, where $E = \text{Set}(V_U(a)) \cup \top$ when $a \in A_{U,\mathbb{1}}$ (see below for the meaning of $\top$) and $E = \text{Set}(\text{Set}(V_U(a))) \cup \top$ when $a \in A_{U,\infty}$. We say that a user u *satisfies* a user attribute expression e_U if and only if, $\forall a \in A_{U,\mathbb{1}}, e_U(a) = \top \vee e_U(a) \ni d_U(u, a)$ and $\forall a \in A_{U,\infty}, e_U(a) = \top \vee \exists s \in e_U(a), d_U(u, a) \supseteq s$. In other words, $\top$ is used to indicate that attribute a is irrelevant for determining whether a user satisfies user attribute expression e_U (i.e., $e_U(a) = \top$).

Resource attribute expressions are defined similarly, except that the satisfaction criterion for multi-valued attributes requires equality rather than $\supseteq$ (i.e., $\exists s \in e(a), d_R(r, a) = s$). The reason is because user attributes which are multi-valued represent capabilities.

A *constraint* represents a relationship between users and resources which may or may not be satisfied, as follows. A constraint c is a function $c : A_U \times A_R \to \{\neg\top, \top\}$. A pair composed of a user u and a resource r satisfies a constraint c if and only if $\forall a_U \in A_{U,\infty}, a_R \in A_{R,\infty}, c(a_U, a_R) = \top \vee d_U(u, a_U) \supseteq d_R(r, a_R)$ and $\forall a_U \in A_{U,\infty}, a_R \in A_{R,\mathbb{1}}, c(a_U, a_R) = \top \vee d_U(u, a_U) \ni d_R(r, a_R)$ and $\forall a_U \in A_{U,\mathbb{1}}, a_R \in A_{R,\mathbb{1}}, c(a_U, a_R) = \top \vee d_U(u, a_U) = d_R(r, a_R)$.

A *rule* ρ is a tuple $\langle e_U, e_R, O, c\rangle$, where e_U is an attribute expression, e_R is a resource expression, c is a constraint and $O \subseteq \mathcal{O}$ is a set of *operations*. A *policy* P is a set of rules. Finally, an *access request* is a tuple $\langle u, r, o\rangle$ which means that user u wants to perform the operation $o \in \mathcal{O}$ on the resource r.

An access request $\langle u, r, o\rangle$ is either *accepted* or *denied* by a rule $\rho = \langle e_U, e_R, O, c\rangle$. The former occurs if and only if u satisfies e_U, r satisfies e_R, u, r satisfies c and $o \in O$. An access request is accepted by a policy P if and only if the access request is accepted by at least one rule in P, otherwise the access request is denied by P. We denote with $\langle u, r, o\rangle \models \rho$ and $\langle u, r, o\rangle \models P$ the acceptance of a request $\langle u, r, o\rangle$ by a rule ρ or a policy P, respectively.

We describe attribute expressions and rules by means of the concrete syntax proposed in [5] and outlined in the following example. Let us consider an university domain in which $A_{U,\mathbb{1}} = \{\text{uid}, \text{position}, \text{isDean}\}$, $A_{U,\infty} = \{\text{courses}\}$, $A_{R,\mathbb{1}} = \{\text{rid}, \text{type}, \text{course}\}$, $A_{R,\infty} = \emptyset$ and $\mathcal{O} = \{\text{writeGrade}, \text{readGrade}, \text{deploy}\}$. A policy P may be composed of the following 4 rules:

$$\rho_1 = \langle \text{position} = \text{student}, \text{type} = \text{gradebook}, \{\text{readGrade}\}, \text{courses} \ni \text{course}\rangle$$
$$\rho_2 = \langle \text{position} = \text{faculty}, \text{type} = \text{gradebook}, \{\text{writeGrade}, \text{readGrade}\}, \text{courses} \ni \text{course}\rangle$$
$$\rho_3 = \langle \text{position} \in \text{faculty} \wedge \text{isDean} = \text{true}, \text{type} = \{\text{gradebook}\}, \{\text{readGrade}\}, \emptyset\rangle$$
$$\rho_4 = \langle \text{courses} \supseteq \{\{\text{CS04}\}, \{\text{WD01}\}\}, \text{type} = \{\text{testWebServer}\}, \{\text{deploy}\}, \emptyset\rangle$$

Rule ρ_1 says that students can read the grades they got for their courses (i.e., those they attend); ρ_2 says that faculty members can read and write grades for their courses (i.e., those they teach); ρ_3 says that the dean can read all grades; ρ_4 says that users whose courses include one among CS04 and WD01 can deploy on the test web server. Note that ρ_1 and ρ_2 pose a constraint on the relationship between the user and the resource, whereas ρ_3 and ρ_4 do not (i.e., for ρ_3 and ρ_4, $c(a_U, a_R) = \top, \forall a_U \in A_U, a_R \in A_R$).

3.2 Problem statement

Let us consider two sets A_U and A_R of user and resource attributes along with their possible values V_U and V_R, and the set $\mathcal{O}$ of operations which may be applied to resources. The problem which we aim to solve consists in generating a policy P which accepts all access requests in a specified set S_A and denies all access requests in another specified set S_D. In other words, the problem consists in inferring a policy *consistent* with specified examples of the desired behavior. A *problem instance* is a tuple $\langle S_A, S_D, A_U, A_R, V_U, V_R, d_U, d_R, \mathcal{O}\rangle$.

A trivial solution for every problem instance always exists in the form of an Access Control List (ACL) policy. Such a policy may be constructed by generating, for each request $\langle u, r, o\rangle \in S_A$, a rule $\rho = \langle e_U, e_R, O, c\rangle$ which accepts only a request from user u to perform the operation o to resource r, using only special attributes uid and rid. In other words, $e_U(a) = d_U(u, a)$ if $a = \text{uid}$, $e_U(a) = \top$ otherwise; $e_R(a) = d_R(r, a)$ if $a = \text{rid}$, $e_R(a) = \top$ otherwise; $O = \{o\}$; and $c(a_U, a_R) = \top$.

In order to generate policies which not only are consistent but indeed generalize beyond the provided examples by taking user and resource attributes into account, we add two further requirements: (i) policy rules should use uid and rid special attributes as little as possible and (ii) the complexity of the policy should be minimized. We assess the complexity of a policy P with the weighted structural complexity (WSC) [8]. WSC is a weighted sums of the complexity of rule components (e_U, e_R, O and c)—see [5] for the details. In this paper we used, without loss of generality, equal weights.

4 Our evolutionary approach

4.1 Overview

We propose an evolutionary approach for solving the policy generation problem. Each individual represents a rule and we define custom genetic operators which operate on rules and are guaranteed to generate valid rules. In other words, we define a domain-specific phenotypic representation of candidate solutions rather than adopting more general representations which would hardly fit this application domain (e.g., trees as in Genetic Programming or numeric vectors as in Genetic Algorithm).

We construct the required policy *incrementally*, by means of successive iterations—a form of separate-and-conquer [16]. At each iteration we execute an evolutionary search which generates one rule $\rho^\star$ and then we drop from the set S_A of requests to be accepted those which are accepted by $\rho^\star$. Each iteration thus operates on a problem instance which differs from the problem instance at the previous iteration—S_A at the $(i+1)$-th iteration being a subset of S_A at the i-th iteration. The procedure terminates when S_A is empty. An intermediate policy is then constructed as the set of rules generated at each iteration. Finally, the required policy is obtained from a further optimization applied to the intermediate policy.

The evolutionary search includes further key contributions:

- We initialize the population based on the problem instance (in particular using the requests in S_A), rather than generating random individuals.
- We promote population diversity by imposing that no identical individuals can be contained in the population.
- We use an *early termination* criterion based on counting how many times the search would attempt to generate an individual which already exists.

4.2 Evolutionary search

An evolutionary search takes a problem instance $\langle S_A, S_D, A_U, A_R, V_U, V_R, d_U, d_R, \mathcal{O}\rangle$ as input and produces a single rule $\rho^\star$. Each individual ρ is associated with a counter c_ρ, initially set to 1, and with a *fitness* $f(\rho)$, defined below. First, an initial population of $|S_A|$ individuals (i.e., rules) is built. These individuals are

generated from S_A rather than randomly, as follows. For each request $\langle u, r, o\rangle \in S_A$ a rule $\rho = \langle e_U, e_R, O, c\rangle$ is built such that:

$$e_U(a_U) = \begin{cases} d_U(u, a_U) & \text{if } a_U \neq \text{uid} \wedge d_U(u, a_U) \neq \bot \wedge \forall a_R \in A_R, c(a_U, a_R) = \top \\ \top & \text{otherwise} \end{cases}$$

$$e_R(a_R) = \begin{cases} d_R(r, a_R) & \text{if } a_R \neq \text{rid} \wedge d_R(r, a_R) \neq \bot \wedge \forall a_U \in A_U, c(a_U, a_R) = \top \\ \top & \text{otherwise} \end{cases}$$

$$O = \{o\}$$

$$c(a_U, a_R) = \begin{cases} \neg\top & \text{if } d_U(u, a_U) \neq \bot \wedge d_R(r, a_R) \neq \bot \wedge d_U(u, a_U) \supseteq d_R(r, a_R) \\ \neg\top & \text{else if } d_U(u, a_U) \neq \bot \wedge d_R(r, a_R) \neq \bot \wedge d_U(u, a_U) \ni d_R(r, a_R) \\ \neg\top & \text{else if } d_U(u, a_U) \neq \bot \wedge d_R(r, a_R) \neq \bot \wedge d_U(u, a_U) = d_R(r, a_R) \\ \top & \text{otherwise} \end{cases}$$

In practice, in order to build a rule $\rho = \langle e_U, e_R, O, c\rangle$ from the request $\langle u, r, o\rangle$ we first find the user and resource attributes which can be used to define the constraint c; then, we set user and attribute expression e_U and e_R according to the values of the respective u and r attributes; in doing so, we consider only attributes which have not been used for defining c and we do not use either uid or rid.

With reference to the university domain example in Section 3.1, let us consider the following request in S_A:

$$u = \langle \text{uid} = \text{stud111013}, \text{position} = \text{student}, \text{courses} = \{\text{CS01}, \text{CS03}\}\rangle$$
$$r = \langle \text{rid} = \text{gradebook7211}, \text{type} = \text{gradebook}, \text{course} = \text{CS03}\rangle$$
$$o = \text{readGrade}$$

The rule generated from this request will be:

$$\rho = \langle \text{position} = \text{student}, \text{type} = \text{gradebook}, \{\text{readGrade}\}, \text{courses} \ni \text{course}\rangle$$

Note that $c(\text{courses}, \text{course}) = \neg\top$ because $d_U(u, \text{courses}) = \{\text{CS01}, \text{CS03}\} \ni \text{CS03} = d_R(r, \text{course})$.

Having generated the initial population, we execute the following iterative procedure:

1. Choose randomly whether to apply a *mutation* operator or a *crossover* operator; the choice between the two options is made with probability p_{mutation} and $1 - p_{\text{mutation}}$, respectively.
2. Choose randomly the specific operator within the chosen category with uniform probability—we defined 10 mutation operators and 5 crossover operators.
3. If a mutation operator has been chosen, then select one rule in the current population, otherwise (a crossover operation has been chosen) select two rules; each rule selection is made by picking $n_{\text{tournament}}$ rules at random and then selecting the best one.

4. Generate a new rule ρ' with the chosen genetic operator applied to the chosen rule(s); if the current population does not already contain a rule $\rho = \rho'$, then add ρ' to the current population and evaluate its fitness $f(\rho)$; otherwise, increment counter c_ρ by one and discard ρ'.
5. If the current population size is greater than n_{pop}, then iteratively remove the worst rule until the population size is equal to n_{pop}.

The iterative procedure terminates when one of the following holds: (a) a predefined number n_{eval} of fitness evaluations has been performed, or (b) the counter $c_{\rho^\star}$ of the best rule $\rho^\star$ is larger than a predefined number n_{stop}. At the end, the best rule $\rho^\star$ in the current population is the result of the search. Note that the fitness of generated rules are evaluated only when they are different from all existing—and hence already evaluated—rules (step 4).

We defined 10 mutation operators and 5 crossover operators. Their full description is not included in this paper for space constraints but is available separately[3]. For example, we defined a constraint donation crossover operator as follows: let $\rho_1 = \langle e_{U,1}, e_{R,1}, O_1, c_1 \rangle$ and $\rho_2 = \langle e_{U,2}, e_{R,2}, O_2, c_2 \rangle$ be the parent rules. The rule $\rho = \langle e_U, e_R, O, c \rangle$ generated by the operator is initially set to $\rho = \rho_1$; next, a pair $a_U, a_R \in A_U \times A_R$ is randomly chosen such that $c_1(a_U, a_R) = \top \wedge c_2(a_U, a_R) = \neg\top$; finally, $c(a_U, a_R) := c_2(a_U, a_R)$.

The fitness $f(\rho)$ of a rule ρ is defined as a tuple composed of 4 numbers: $f(\rho) = \langle \text{FAR}(\rho), \text{FRR}(\rho), \text{ID}(\rho), \text{WSC}(\rho) \rangle$, where FAR and FRR are the False Acceptance Rate on the requests in S_D and the False Rejection Rate on the requests in S_A, respectively; $\text{ID}(\rho)$ is a measure of the usage of the special attributes uid and rid; $\text{WSC}(\rho)$ is the WSC index defined in Section 3.2. In detail:

$$\text{FAR}(\rho) = \frac{|\{\langle u, r, o \rangle \in S_D, \langle u, r, o \rangle \models \rho\}|}{|S_D|}$$

$$\text{FRR}(\rho) = \frac{|\{\langle u, r, o \rangle \in S_A, \langle u, r, o \rangle \not\models \rho\}|}{|S_A|}$$

$$\text{ID}(\rho) = \begin{cases} 2 & \text{if } e_U(\text{uid}) \neq \top \wedge e_R(\text{rid}) \neq \top \\ 1 & \text{if } e_U(\text{uid}) \neq \top \veebar e_R(\text{rid}) \neq \top \\ 0 & \text{otherwise} \end{cases}$$

For all the elements of the fitness tuple, the lower the better.

Rules are ranked basing on lexicographical order of their fitnesses $f(\rho)$: the rule with lower FAR is considered the best; in case two or more have the same lowest FAR, the rule with lowest FRR is considered the best; in case two or more have the same lowest FRR, the rule with lowest ID is considered the best; in case two or more have the same lowest ID, the rule with lowest WSC is considered the best. This method of ranking solutions in a multi-objective problem where objectives are sorted by decreasing importance is also known as *multi-layered fitness* [17]. In our case, in the fitness $f(\rho) = \langle \text{FAR}(\rho), \text{FRR}(\rho), \text{ID}(\rho), \text{WSC}(\rho) \rangle$,

[3] http://machinelearning.inginf.units.it/data-and-tools/appendices/2014-EMO-EvolutionaryABACInference-Appendix.pdf

the first two components represent the ability of the rule to be consistent with the problem instance, whereas the other two reflect the further problem objectives concerning use of special attributes and complexity (see Section 3.2).

4.3 Incremental strategy

We construct the required policy incrementally, by means of successive evolutionary searches, as follows. Initially, let $P = \emptyset$ and $S'_A = S_A$, then:

1. execute an evolutionary search (Section 4.2) on problem instance $\langle S'_A, S_D, A_U, A_R, V_U, V_R, d_U, d_R, \mathcal{O} \rangle$ and obtain $\rho^\star$;
2. if $\mathrm{FAR}(\rho^\star) = 0$ and $\mathrm{FRR}(\rho^\star) < 1$ then $P := P \cup \{\rho^\star\}$, otherwise terminate;
3. assign $S'_A = S_A \setminus \{\langle u, r, o \rangle \in S_A, \langle u, r, o \rangle \models P\}$;
4. if $S'_A = \emptyset$, terminate.

In other words, at each iteration we obtain a new rule $\rho^\star$ (step 1). As long as this new rule accepts at least one request in S'_A (step 2, $\mathrm{FRR}(\rho^\star) < 1$) while not accepting any request in S_D ($\mathrm{FAR}(\rho^\star) = 0$), the new rule is added to the policy being constructed and the iteration continues. The next iteration will operate on a smaller S'_A, which does not contains any request accepted by the current policy, including $\rho^\star$ (step 3). In case all requests to be accepted are already accepted by the current policy, the iteration terminates (step 4).

Since each $\rho \in P$ has $\mathrm{FAR}(\rho) = 0$—see step 2 above—and since a request is accepted if at least one rule in P accepts it, it follows that $\mathrm{FAR}(P) = 0$ and $\forall \rho \in P, \mathrm{FRR}(P) \leq \mathrm{FRR}(\rho)$, where FRR and FAR are defined for policy P similarly to for requests.

The (intermediate) policy P obtained by the above procedure is optimized further by executing the following procedure for a predefined number of n_{eval} iterations:

1. choose a rule ρ in P at random;
2. generate a new rule ρ' by applying a randomly selected mutation operator on ρ;
3. build a policy P' by replacing ρ with ρ' in P;
4. if P' is better than P, than $P := P'$.

The comparison criterion between two policies P_1, P_2 is based on the same lexicographical order used for rules: the policy with lowest FAR is considered the best; otherwise, the policy with lowest FRR is best; otherwise, the policy with lowest ID is best (where $\mathrm{ID}(P) = \sum_{\rho \in P} \mathrm{ID}(\rho)$); otherwise, the policy with lowest WSC is best.

We chose to use a lexicographical order (both for rules and policies) because it reflects the order of in which the problem objectives are defined (consistency first, then use of special attributes, then complexity). Moreover, concerning consistency, we chose to favor—i.e., minimizing first—FAR instead of FRR because of the way we compose a policy starting from rules: in particular, we aim at obtaining rules with $\mathrm{FAR} = 0$ (see the condition in step 2 above).

5 Experimental evaluation

We evaluated our proposal experimentally on the same case studies considered in [5]. Each case study consists of a set of users U, a set of resources R and a set of rules P_0. Users and resources are associated with various attributes. Rules were carefully constructed to express non-trivial policies and exercise all the features of the policy language, including use of set membership and superset relations in attribute expressions and constraints. The experimental data consist of 7 case studies: 4 of them were hand-crafted and 3 of them were synthetically generated from the hand-crafted ones. The synthetic case studies include a much larger number of users and resources. Table 1 summarizes the case studies. The set of operations $\mathcal{O}$ is obtained from P_0 as $\mathcal{O} = \bigcup_{\langle e_U, e_R, O, c\rangle \in P_0} O$. The set of requests includes all possible requests, i.e., $S = U \times R \times \mathcal{O}$; this set is then partitioned in S_A and S_D, basing on whether each request in S was accepted or denied by P_0, respectively.

Case study	$\lvert P_0\rvert$	$\lvert U\rvert$	$\lvert R\rvert$	$\lvert\mathcal{O}\rvert$	$\lvert A_U\rvert$	$\lvert A_R\rvert$	$\lvert S_A\rvert$	$\lvert S_D\rvert$	WSC(P_0)
Healthcare	9	21	16	3	6	7	51	957	33
Online video	6	12	13	1	3	3	78	78	20
Project management	11	19	40	7	8	6	189	5131	49
University	10	22	34	9	6	5	168	6564	37
Healthcare†	9	1600	5760	3	6	7	10 097	27 637 903	33
Project management†	11	800	1600	7	8	6	7680	8 952 320	49
University†	10	1320	2520	9	6	5	148 624	29 788 976	37

Table 1. Salient information about the hand-crafted (above) and synthetic (below, with a † suffix) case studies.

We executed our approach on each case study for several values of the n_{eval} parameter. We repeated each experiment 3 times for each n_{eval} value, with different random seeds. We set the other parameters as follows: $n_{\text{pop}} = 100$, $n_{\text{tournament}} = 3$, $n_{\text{stop}} = 100$ and $p_{\text{mutation}} = 0.5$—we verified experimentally that reasonable variations in these values do not cause significant variations in the results.

Since the synthetic case studies are associated with several millions of requests to be denied, in these cases we generated the corresponding policies based on a random sample S_D^* of those requests such that $|S_D^*| = 5|S_A|$. The results in terms of FRR and FAR have always been computed on the full set, though.

Table 2 presents the results, averaged across the executions with different random seeds, in terms of FRR(P), FAR(P) and $\frac{\text{WSC}(P_0)}{\text{WSC}(P)}$; the table also shows the actual number $\hat{n}_{\text{eval}}$ of fitness evaluations—recall that the number of iterations of the incremental strategy is not known in advance—and the execution time.

The first crucial finding is that our approach indeed succeeds in generating consistent policies, i.e., policies with FRR(P) = 0, FAR(P) = 0. Moreover, we

Case study	n_{eval}	FRR(P)	FAR(P)	$\frac{\text{WSC}(P_0)}{\text{WSC}(P)}$	$\hat{n}_{\text{eval}}$	t [s]
Healthcare	500	0	0	1.07	5536	1.2
	2500	0	0	1.18	19 776	4
	5000	0	0	1.18	22 691	5.3
Online video	500	0	0	1	2768	0.6
	2500	0	0	1	5215	0.8
	5000	0	0	1	7715	1.1
Project management	500	0	0	0.96	6646	3.5
	2500	0	0	1.06	24 368	14.7
	5000	0	0	1.06	27 791	22.2
University	500	0	0	0.95	5904	3.1
	2500	0	0	0.98	22 846	14.1
	5000	0	0	1	26 487	21.8
Healthcare†	500	0	0	1.18	23 704	228.4
	2500	0	0	1.2	33 864	398.9
	5000	0	0	1.2	36 364	511.9
Project management†	500	0	0	0.92	35 037	241.7
	2500	0	0	1.06	45 591	626.9
	5000	0	0	1.06	49 549	790
University†	500	0	0	0.89	35 822	1688.8
	2500	0	0	1	52 513	3525.4
	5000	0	0	1	56 718	4784.9

Table 2. Results.

verified that our method never produced policies which use special attributes uid and rid, as desired.

Another important result is that our approach definitely tends to generate a policy which is less complex than the baseline: in most cases WSC(P) is not larger than the WSC(P_0) (we remark that P_0 is unknown to our approach). This effect is more apparent for Healthcare and Project management and the corresponding synthetic versions, but can be observed also for University and its synthetic counterpart, for sufficiently large values of n_{eval}. In other words, our approach aims at obtaining the least complex policy which is consistent with the desired behavior in terms of S_A, S_D; as it turns out, the generated policy tends to be less redundant than the baseline policy.

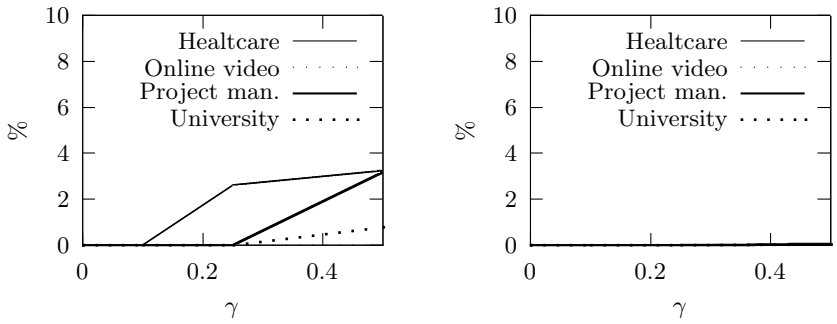
The average execution time for generating a policy is in the order of seconds for the hand-crafted case studies and in the order of minutes or a few tens of minutes for the synthetic ones. It seems reasonable to claim that the computational load is fully practical for this application domain. The experiments have been executed with a single-threaded Java prototype implementation on a quad-core Intel CPU 2.50 GHz with 8 GB RAM. As expected, the execution time is roughly linear with $\hat{n}_{\text{eval}}|S_A||S_D|$; with respect to n_{eval}, it can be seen that time is slightly sublinear due to the intervention of the early termination criterion given by n_{stop}.

5.1 Results with incomplete input

We wanted to gain insights in our method effectiveness when the input information is incomplete. In particular, we considered the case where the input requests sets S_A and S_D do not contain all the requests.

To this end, we repeated the experimental procedure detailed in the previous section, by removing, before applying our method, a random portion γ of requests in S_A and S_D. The performance in terms of FRR and FAR has obviously been computed on full S_A and S_D. We repeated each experiment 3 times—i.e., with 3 different S_A, S_D and random seeds—for each value of γ.

The corresponding results are in Figure 1 (we executed these experiments only on the hand-crafted case studies). It can be seen that the removal of part of requests has little or no impact on FAR, even for large values of γ. Indeed, FAR is always 0 when $\gamma \leq 0.25$ and remains low ($\leq 1\%$) even when half of the requests in S_D are not available for inferring the policy. The impact of the removal of part of requests is slightly higher on FRR, but always lower than 4%. We believe the reason is because our approach tends to produce a policy which contains only the rules which are needed to accept all the requests contained in the input S_A. This interpretation is supported by the values of $\frac{\text{WSC}(P_0)}{\text{WSC}(P)}$, which become larger with large values of γ: in other words, our approach produces the least complex policy which is consistent with the input.



(a) FRR (b) FAR

Fig. 1. Results with incomplete input.

5.2 Assessment of the contributions

We wanted to assess the specific impact of our key contributions described in Section 4.1: population initialization from requests in S_A, incremental strategy, diversity promotion and early termination. Rather than assessing all combinations, we executed pairwise comparisons between the full approach and the

approach with one of these contributions disabled—note that disabling the diversity promotion implies disabling also the early termination. We considered only the hand-crafted case studies.

Concerning the population initialization from the examples in S_A, we experimented with random creation of individuals in the initial population. It was clear from the early experiments that generating a consistent policy from a random initial population is very difficult. For this reason, we investigated the impact of increasing the population size $n_{\text{pop}} = 100$ by an order of magnitude $(1000, 2000)$ and disabled early termination. Table 3 shows the results obtained with random population initialization (for ease of comparison we provide the corresponding results of the full method previously shown in Table 2). It is clear that the population initialization from S_A is an essential ingredient for generating consistent policies and greatly improves the method effectiveness. Table 3 shows that, if one favors the exploration ability of the evolutionary search by increasing n_{pop} and n_{eval}, the method can indeed obtain better solutions also with random initialization. These improved results are still far from those with the initialization from S_A, though, despite the increased execution time.

			Healthcare		Online video		Proj. man.		University	
Pop. init.	n_{eval}	n_{pop}	FRR(P)	t[s]	FRR(P)	t[s]	FRR(P)	t[s]	FRR(P)	t[s]
From S_A	5000	100	0	5.3	0	1.1	0	22.2	0	21.8
Random	5000	100	37.3	5.5	3.4	7.1	56.6	11.4	73.4	9.1
	25 000	100	32.7	28.2	1.7	37.2	53.1	61.8	69.4	49.1
	25 000	1000	5.2	203.7	0	300.8	38.4	224.7	9.5	276.5
	25 000	2000	2.6	432.1	0	642.0	34.9	394.1	36.1	387.9

Table 3. Results with and without the population initialization from S_A.

Concerning the incremental strategy, we executed a single evolutionary search followed by the final optimization step executed with $n_{\text{eval}} = 5000$—results with larger values for n_{eval} are essentially the same. The results in terms of FRR are significantly worse than with the incremental strategy enabled: 82.4%, 53.9%, 83.1% and 74.6% for the Healthcare, Online video, Project management and University case studies respectively, as opposed to FRR = 0. In other words, a single evolutionary search is not able to generate a rule capable of accepting all the requests in S_A and denying all the requests in S_D.

Concerning diversity promotion, we experimented without this contribution and $n_{\text{eval}} = 5000$. We found that our approach still generates consistent policies (i.e., with FAR = FRR = 0), but with larger complexity: $\frac{\text{WSC}(P_0)}{\text{WSC}(P)}$ is lower than with the diversity promotion for the Healthcare and Project management case studies: 1.15 vs. 1.18 and 1.01 vs. 1.06, respectively.

Concerning the early termination criterion, we experimented by stopping the evolutionary search after having performed $n_{\text{eval}} = 5000$ fitness evaluations irrespective of the number of times the search attempts to generate identical individuals. The quality of the results is unaffected, the only impact being on

execution times which are, on the average, 128% longer than with early termination enabled.

Finally, we investigated on the effectiveness of the further optimization of the policy P which we perform at the end of the incremental strategy (see Section 4.3). To this end, we experimented without this step. We found that this procedure impacts on the complexity of the generated policies: $\frac{\mathrm{WSC}(P_0)}{\mathrm{WSC}(P)}$ is equal to 1.18, 0.91, 1.04 and 0.98 for the Healthcare, Online video, Project management and University case studies respectively, as opposed to 1.18, 1, 1.06 and 1, with the optimization. By looking at the raw data, we found that the optimization makes our approach cope with those cases where a rule ρ_1, which has been generated at a given iteration of the incremental strategy, could be made less complex because of a rule ρ_2 generated later, which accepts some requests accepted also by ρ_1.

6 Concluding remarks and future work

We have proposed an evolutionary approach for performing mining of ABAC policies. The approach is based on the design and implementation of a domain-specific phenotypic representation, along with the corresponding genetic operators, which allow attacking ABAC policy mining by means of evolutionary computation. We used a multi-objective optimization framework based on a lexicographic criterion, in which we incorporate requirements on correctness (FAR, FRR) and on expressiveness (WSC, usage of uid and rid). We incorporated in the search several optimizations that have proven to be essential for this task, in particular, we defined a strategy for building a policy incrementally, by learning single rules each one on a different subset of the requests. We showed that our approach deals successfully with case studies of realistic complexity, being highly robust even in scenarios where the access requests available for learning do not fully represent the access control information. We believe that our proposal may indeed form the basis for a practical implementation of ABAC policy mining and we intend to extend the scope of our investigation to emergency management.

Acknowledgment:

The research presented in this paper is partially supported by the project *Dynamic Information Management and Exchange for Command and Control Applications*, grant number FA8655-10-1-3080, funded by the European Office of Aerospace Research and Development (EOARD) and the Air Force Office of Scientific Research (AFOSR).

References

1. Ferrari, E.: Access Control in Data Management Systems. Synthesis Lectures on Data Management. Morgan & Claypool Publishers (2010)

2. Hu, V.C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R., Scarfo, K.: Guide to Attribute Based Access Control (ABAC) Definition and Considerations. In: NIST Special Publication (SP) 800-162, Guide. (Oct. 2014)
3. Brucker, A.D., Petritsch, H.: Extending access control models with break-glass. In: Proceedings of the 14th ACM symposium on Access control models and technologies, ACM (2009) 197–206
4. Carminati, B., Ferrari, E., Guglielmi, M.: A System for Timely and Controlled Information Sharing in Emergency Situations. Dependable and Secure Computing, IEEE Transactions on **10**(3) (2013) 129–142
5. Xu, Z., Stoller, S.D.: Mining attribute-based access control policies. arXiv preprint arXiv:1306.2401 (2013)
6. Xu, Z., Stoller, S.D.: Mining attribute-based access control policies from RBAC policies. In: Emerging Technologies for a Smarter World (CEWIT), 2013 10th International Conference and Expo on, IEEE (2013) 1–6
7. Gal-Oz, N., Gonen, Y., Yahalom, R., Gudes, E., Rozenberg, B., Shmueli, E.: Mining roles from web application usage patterns. In: Trust, Privacy and Security in Digital Business. Springer (2011) 125–137
8. Molloy, I., Chen, H., Li, T., Wang, Q., Li, N., Bertino, E., Calo, S., Lobo, J.: Mining roles with multiple objectives. ACM Trans. Inf. Syst. Secur. **13**(4) (December 2010) 36:1–36:35
9. Ni, Q., Lobo, J., Calo, S., Rohatgi, P., Bertino, E.: Automating role-based provisioning by learning from examples. In: Proceedings of the 14th ACM symposium on Access control models and technologies, ACM (2009) 75–84
10. Hu, N., Bradford, P.G., Liu, J.: Applying role based access control and genetic algorithms to insider threat detection. In: Proceedings of the 44th annual Southeast regional conference, ACM (2006) 790–791
11. Lim, Y.T., Cheng, P.C., Rohatgi, P., Clark, J.A.: MLS security policy evolution with genetic programming. In: Proceedings of the 10th annual conference on Genetic and evolutionary computation, ACM (2008) 1571–1578
12. Lim, Y.T., Cheng, P.C., Rohatgi, P., Clark, J.A.: Dynamic security policy learning. In: Proceedings of the first ACM workshop on Information security governance, ACM (2009) 39–48
13. Bleuler, S., Brack, M., Thiele, L., Zitzler, E.: Multiobjective genetic programming: Reducing bloat using SPEA2. In: Evolutionary Computation, 2001. Proceedings of the 2001 Congress on. Volume 1., IEEE (2001) 536–543
14. Tapiador, J.E., Clark, J.A.: Learning autonomic security reconfiguration policies. In: Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on, IEEE (2010) 902–909
15. Bartoli, A., Cumar, S., De Lorenzo, A., Medvet, E.: Compressing Regular Expression Sets for Deep Packet Inspection. In: Parallel Problem Solving from Nature–PPSN XIII. Springer (2014) 394–403
16. Fürnkranz, J.: Separate-and-conquer rule learning. Artificial Intelligence Review **13**(1) (1999) 3–54
17. Eggermont, J., Kok, J.N., Kosters, W.A.: Genetic programming for data classification: Partitioning the search space. In: Proceedings of the 2004 ACM symposium on Applied computing, ACM (2004) 1001–1005